P DE1[illegible]

NASA-TM-112264

7N-64-TM
[illegible]

Natural Manipulation of Surfaces via Spatially Weighted Transformations

Steve Bryson

RNR Technical Report RNR-91-004, January 1991

NASA
National Aeronautics and
Space Administration

Ames Research Center
Moffett Field, California 94035

ARC 275 (Rev Feb 81)

Natural Manipulation of Surfaces via Spatially Weighted Transformations

Steve Bryson

RNR Technical Report RNR-91-004, January 1991

Computer Sciences Corporation/
Applied Research Office, Numerical Aerodynamics Simulation Division
NASA Ames Research Center
MS T045-1
Moffett Field, Ca. 94035
bryson@nas.nasa.gov

Abstract

This paper describes a method of directly manipulating a surface in computer graphics. The manipulation is natural in the sense that when a point on the surface is moved via some transformation, other points on the surface follow the motion in a way reminiscent of taffy or putty. Points near the point transformed move almost as much while points far away do not move at all. This is accomplished by defining a 'bump' weight function on the surface which is shaped and placed by the user. After the manipulation, the vertices of the surface are replaced by their transformed images. In this way the user can naturally 'sculpt' the surface by simply moving parts of it around. This system generalizes to any transformation and any dimensionality. A simple implementation is described in detail.

Keywords: free-form surfaces, deformations, solid modeling, interaction techniques, direct manipulation.

1: Introduction

1.1: Purpose and Motivation

The sculpting and manipulation of surfaces in computer graphics is a well-studied problem. The interactive definition of surfaces has been addressed in many ways including splines and free form deformations [2, 4, 7, 8, 14, 15, 17]. The ability to 'reach out and grab' a part of a surface for direct manipulation is the paradigm which will be developed in this paper. The concept developed in this paper is that as a point p on a surface is 'grabbed and moved', points near p move almost as much as p, while points far from p (perhaps) do not move at all (Fig 1). In this way the surface can be treated as a stretchable material, much like putty or melted cheese. Thus a user would be able to sculpt a shape out of a surface by moving parts of the surface where he or she wants them (fig. 2). 'Grabbing a point' means indicating the point on the surface which is to be the center of the movement. "Moving the point" means that the coordinates of that point are mapped to some other values through some transformation T. In this paper it is assumed that there is some way of indicating the point to be grabbed, and of defining the transformation. This paper describes a method of determining the motion of all the other points on a surface, based on the movement of the grabbed point and parameters specified by the user.

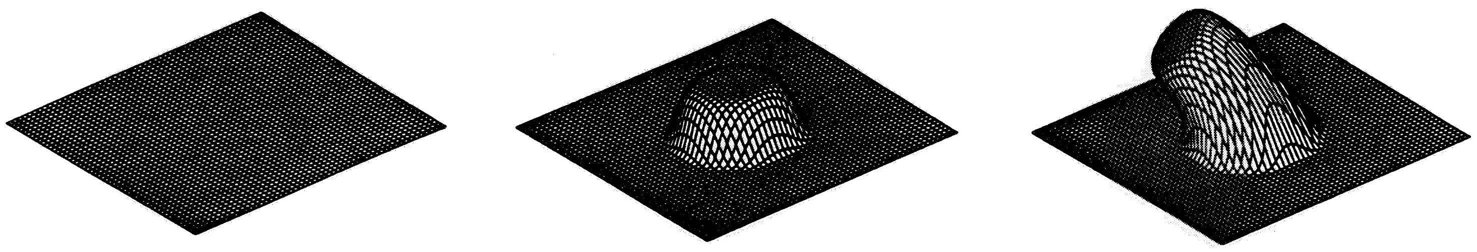


Fig 1: Three stages in the manipulation of a surface, involving both translation and rotation.

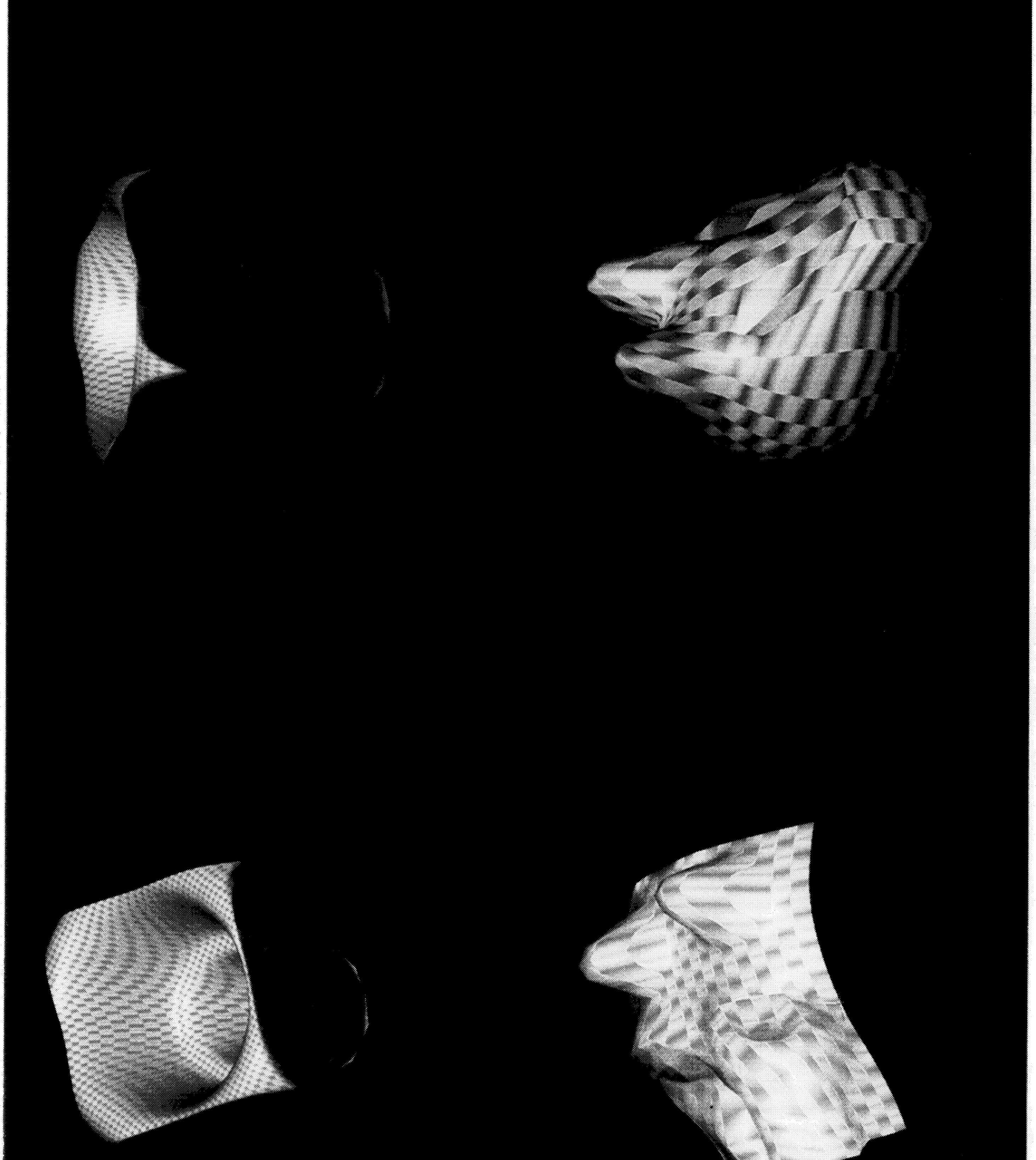
To be useful, this method must give the user the capability of easily specifying:
- which points move with precisely the motion specified by the user
- which points on the surface do not move at all (so that moves can be independent)
- how the remaining points (which are between the two cases above) move.

In addition, the current specification of these parameters should be easy to understand and visualize.

While the basic idea of the points moving near the point grabbed is an obvious one, the user may also wish for more exotic and non-local possibilities. These may include grabbing the entire edge of a surface, moving a ring on the surface around the point grabbed (without moving the center of the ring), or moving the entire surface as a rigid body. In general, there may be situations where moving an arbitrary subset of the surface is desired. The method described in this paper allows in principle for all of these possibilities, and a limited implementation with many of these features will be described.

In addition to sculpting, the method developed in this paper may be useful in the field of topological visualization [9]. In this field, it is desired that the user be able to manipulate a surface under certain rules to investigate the topological properties of that surface. There are methods of topological reasoning, i.e. surgery [13] and handlebody theory [11], that are very visual but have no interactive tools other than hand drawings. The development of a surface manipulation system specifically for topological visualization was the original motivation for the work described in this paper. For this reason, it is desirable to have the manipulation of the surface preserve the topology of that surface (roughly, how the vertices of the surface are connected).

Fig 2: Examples of surfaces sculpted using the method described in this paper. Clockwise from upper left: two views of a cup; a boar's head; a cat mask.



While the method described in this paper is discussed in terms of the manipulation of two-dimensional surfaces in three-dimensional space, the generalization to higher (or lower) dimensions is straightforward.

1.2 Brief Survey of Existing Solutions to the Manipulation Problem

The most well developed approach to the interactive manipulation of surfaces in computer graphics is based on cubic interpolation through defining points. This concept has developed into the well known spline approach and its variants [2, 7, 8], which use control points to determine the shape of the surface. These control points may or may not actually be on the surface to be manipulated. The spline approach has the advantage that relatively few points can be used to define a surface, but the disadvantage that when the control points are moved, the changes in the surface do not precisely track the motion of the control point. Instead the surface changes in highly constrained and at times non-local ways (depending on the type of spline used). This problem has been addressed with the concept of hierarchical spline systems [8]. Using splines, it is difficult to change from, for example, highly localized to very broad manipulations at a given point. This difficulty is because the control points actually define the surface, not the parameters of the manipulation of the surface. Thus changing the behavior of the control points do requires a redefinition of the surface. To avoid these problems, the method described in this paper separates the parameters of the manipulation of the surface from the definition of the surface. Also, in the more versatile and advanced spline methods the manipulation is via control points that are off the surface: the surface itself is not grabbed, and knowing which control point is appropriate for the desired motion is not obvious. Finally, the generalization to higher dimensional splines is simple in principle but technically non-trivial.

Another, more recent, method of surface manipulation is that of free-form deformation [15, 4], which assigns a three-dimensional lattice to a volume enclosing the object to be deformed. The coordinates of the object are computed with respect to the lattice. The lattice is then deformed through the manipulation of its vertices, which produces deformed coordinates at each lattice vertex. The coordinates of the vertices in the object in lattice coordinates are then composed with the coordinates of the deformed lattice vertices to produced a distortion of the object. This method has several advantages: the distinction between the object definition and the distortion definition; the ability to produce both local and global deformations by dynamic choice of the lattice; and the trivial generalization to arbitrary dimensions. In this system, however, it is the lattice points, not the surface itself that is 'grabbed'. Also, only those distortions that are allowed by available lattices are possible [4]. More seriously, if pieces of a surface are folded over on each other, one can encounter situations where two parts of the surface are in the same lattice cell, making it difficult to move them apart again. Finally, even if the lattice for the desired manipulation is available, the user must know which one it is and select it, thus entailing some extra work.

A surface manipulation system based on the deformable surfaces developed by Terzopoulos et.al. [16] has been implemented by Waters and Wang [17]. This system enables the direct manipulation of surfaces, and the problem of the motion of neighboring vertices is solved through treating the surface as some physical medium. While Waters and Wang's approach shares many intuitions with the system developed in this paper, the physical simulation underlying their approach would constrain the types of manipulations allowed. Further, the computational task involved in computing the physical simulations constrains the real-time capabilities of an implementation.

A completely different approach to the deformation of surfaces and solid objects is that of Barr [1], where objects are deformed through linear transformations that are weighted with a function of time or position. While Barr did not suggest a method of manipulation, his underlying principles are very similar to those described in this paper.

As part of a general system for sculpting 3D data, Parent [14] described a simple scheme which is also similar to the method developed in this paper. Parent presents an interpolation scheme in which the movement of a point on a surface causes nearby points to move as if they are elastically attached to the moved point. This scheme has the beginnings of the system developed in this paper, but is limited to the movement of individual points and uses a non-intuitive weighting function.

In a sense, the system described in this paper is a combination and generalization of both Barr's and Parent's work in such a way that an interface is easy to construct.

1.3 Brief Description of the Algorithm

The purpose of the system described in this paper is to allow the user to 'reach out' and 'grab' a virtual surface with an input device, and as the user moves the input device:

a) some specified region of the surface moves precisely as indicated by that device,

b) some other specified region of the surface does not move at all

and

c) the rest of the surface nicely interpolates between these two regions.

This system treats the surface as a collection of vertices with connectedness, and defines a transformation T on these vertices via some input device. A smooth mask m is generated which is equal to 1 in some specified region where the vertices will be transformed by T, 0 in some other specified region where the vertices will not be transformed at all, and some number between 0 and 1 everywhere else, so that these vertices will be transformed by m*T.

In effect, we are defining a weighting function m(v) on each vertex v such that $0 \leq m(v) \leq 1$. Further, this weighting function is to be continuous in the sense that m(v) = m(s, t) where s and t are parameters on the surface at the vertex v and m(s, t) is continuous, and that derivatives to some specified order exist and are continuous. In fact, a weight function which has continuous derivatives to all orders (i. e. is smooth) everywhere except for a one-dimensional curve (where the second derivative is discontinuous) will be used. This means that in practice, since the probability of a vertex falling on this curve is zero, the transformations on the vertices will be smooth everywhere. While many other weight functions that are less smooth will be similarly well behaved in practice, we have chosen one that a) has a nice shape, b) is almost smooth, and c) is not too time consuming to compute (A similar function that is smooth everywhere is defined in terms of integrals which must be evaluated numerically).

When the surface is 'grabbed' and manipulated, only the part of the surface grabbed should move, and not any overlapping or intersecting parts of the surface. This requires that we define the continuity of the weight function with respect to distances along the surface, not the distance between vertices in three dimensions. More specifically, we are demanding that the weight function is local with respect to the surface, not with respect to the space the surface is immersed in. This way if, for example, the surface self-intersects we can manipulate one part of the surface at the intersection without moving the intersecting part.

Once a single manipulation of the surface has been performed, the vertices defining the surface are replaced by their images under the masked transform described above, producing a new surface which may again be manipulated. In this sense, this system is iterative: the surface is replaced by its manipulated image.

2: Theory

2.1 Definition of the bump weighting function

A weighting function is needed that is constant and equal to one in some set of numbers, is constant and equal to 0 outside some larger set of numbers that contains the first one, is monotonically decreasing from 1 to 0 in between, and is smooth. A function which is versatile, easy to control, and has a nice shape is the bump function.

The construction of this function is based on the function

$$f(x) = \begin{cases} 0 & x \leq 0 \\ e^{-\left(x^{-2}\right)} & x > 0 \end{cases}$$

f(x) has well-defined derivatives to all orders at 0 (its derivatives are of the form (polynomial in x)*(polynomial in (1/x))*f(x), and f(x) will go to zero faster than the polynomial in (1/x) diverges as x goes to zero) [10, exercise 1.1.18]. This function is used to construct the smooth step function

$$s(x, a) = \begin{cases} 0 & x \leq 0 \\ e^{-\left(\frac{x}{a-x}\right)^{-2}} & 0 < x < a \\ 1 & x \geq a \end{cases}$$

which has continuous derivatives to all orders at x=0 (fig. 3). At x=a, where a is any positive number, this function has discontinuous second and higher derivatives. (In general, the function

$$s_p(x, a, p) = \begin{cases} 0 & x \leq 0 \\ e^{-\left(\frac{x}{a-x}\right)^{-p}} & 0 < x < a \\ 1 & x \geq a \end{cases}$$

will have continuous derivatives up to order p-1 at x=a. Thus one can have higher degrees of continuity at x=a at the cost of steeper bumps.) By translation, the function

$$\text{step}(x, r0, r1) = s(x\text{-}r0, r1\text{-}r0)$$

is a function that is equal to 0 for $x \le r0$, equal to 1 for $x \ge r1$, and is smooth between r0 and r1, where $0 < r0 < r1$. Finally, one can define the bump function, given by

$$\text{bump}(x, c, r0, r1) = \begin{cases} \text{step}(x, c-r0, c-r1) & x < c \\ \text{step}(2c-x, c-r0, c-r1) & x \ge c \end{cases}$$

where c is any number which gives the center of the bump, r1 is the distance from the center within which the bump is equal to 1, and r0 is the distance from the center beyond which the bump is equal to 0 (fig 4).

The function bump(x, c, r0, r1) defines a bump which is controlled by three parameters: c, which is where on the real line the bump sits; r0, the width of the nonzero parts of the bump; and r1, the width of the plateau where the bump is constant and equal to one. Changing c changes the location of the bump. Changing r0 changes the size of the nonzero part of the bump, and finally changing r1 changes the size of the plateau of the bump. The task of controlling the bump is the task of controlling these three parameters.

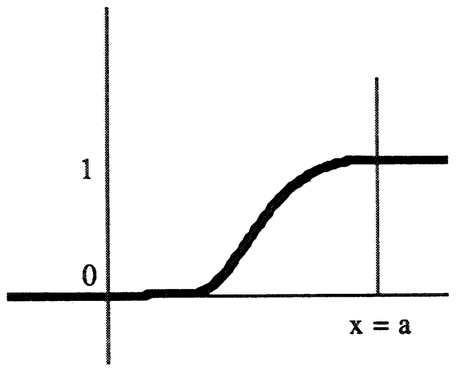


Fig 3: The basic step function s(x, a).

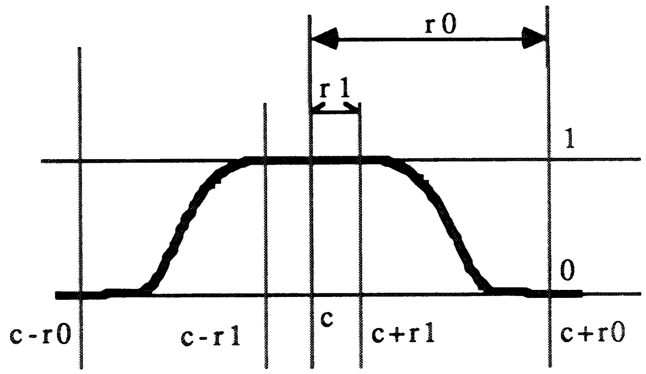


Fig. 4: The bump function, showing the parameters c, r1, r0 which define the bump.

2.2: Definition of the vertex transformation for the local manipulation of a surface.

The bump function defined in section 2.1 is very useful for specifying the way in which a surface is manipulated. Intuitively, when a user specifies that a surface has been grabbed at a particular point p and moves the controller to define some transformation T, then the surface close to p should be fully transformed by T, moving with the controller. The effect of the transformation should fall off gradually with increasing distance from p on the surface, until far away, the surface does not move at all. This is accomplished by using the bump function with x = distance from p and taking c to be zero (the definition of distance and nonzero values of c will be discussed shortly). At each vertex of the surface, the transformation at that vertex is defined as:

$$T'(x) = \text{bump}(x, c, r0, r1)*T + (1-\text{bump}(x, c, r0, r1))*Id$$

where x is the distance of the vertex from p and Id = the identity transformation. This is simply a weighted linear combination of the identity transformation and the transformation T defined through the user's input. Where bump(x, c, r0, r1) = 1, T'(x) = T, and where bump(x, c, r0, r1) = 0, T'(x) = Id. Note that T need not be a linear transformation, but can be any map from three-dimensional space to three-dimensional space.

There is one serious ambiguity in this scheme: the definition of distance on the surface. One could use the euclidean three-dimensional distance between the vertices, but when the surface is folded over on itself, say folded corner to corner, then grabbing one corner will cause the opposite corner to move (See fig. 5). The points in the center of the surface, which we usually call closer to one corner than the opposite corner, will move less, and the effect could be that we have grabbed the two corners at once. While this may be desirable in special circumstances (see the section on topology below), this is not what is intuitively meant by "grabbing the surface at a point".

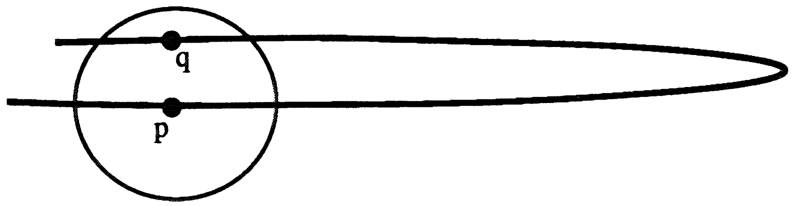


Fig 5: A cross section of a folded surface. The circle shows where the weight function is nonzero if the weight function is a function of the distance between the vertices on the surface. Then if vertex p is grabbed and moved, vertex q will also be moved, even though p is far from q with respect to distance measured on the surface.

The simplest solution is to treat the surface as a parameterized surface, so that the initial undeformed surface is the image of some (preferably smooth) function $\gamma(s, t)$ which, for each value of s and t, gives a point in three-dimensional space. This means that each vertex defining the initial surface is an image of $\gamma(s, t)$ for some particular value of s and t. The parameterization of the surface can be chosen (for convenience) so that the entire surface is the image of the square $0 \leq s \leq 1$ and $0 \leq t \leq 1$ (fig. 6). We can then define the distance between two vertices as the distance in (s, t) space between the two points on the square which get mapped to those vertices. Given a point p on the surface and its corresponding parameters (sp, tp), then a point that is closer than another in (s, t) space will map to a point that is closer on the surface. When the surface is deformed after manipulation, vertices that come from similar values of (s, t) will be closer together than vertices that come from very different values of (s, t), so the weighting function will still be local with respect to the surface.

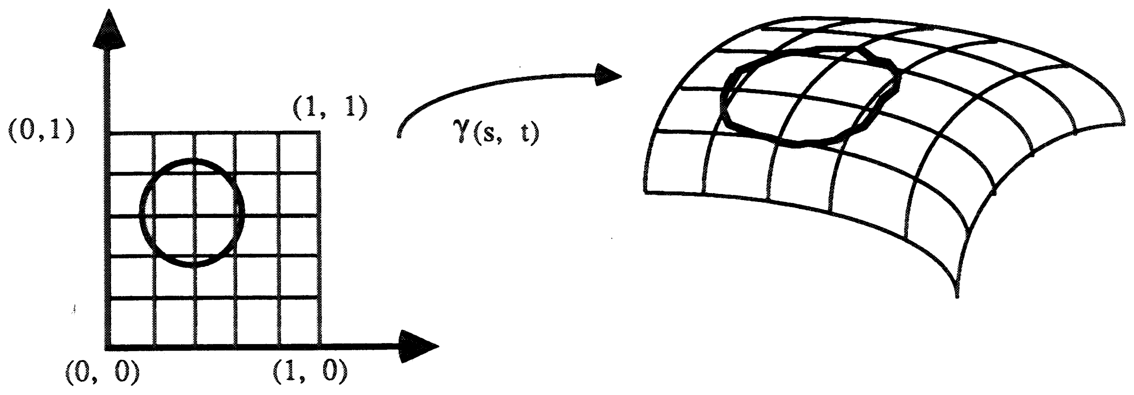


Fig 6: The definition of a parametrized surface as a function from the unit square to three-dimensional space. The image of the circle on the surface represents where the weight function is nonzero, when distance is measured in the (s, t) parameter space.

There is one disadvantage to this definition of distance on the surface. If adjacent vertices on a surface are stretched far apart, they will still be treated as adjacent and therefore near with respect to the bump function. This is because they are images of nearby points in the parameter space. Thus vertices that have been moved far apart can move together if one of them is grabbed. Another way to see this problem is that the bump function, which is circular on the undistorted surface, will be distorted on the distorted surface. Possible solutions to this problem will be briefly discussed in section 4. In the current implementation of this system, an interface (which is discussed in section 3.3) was developed that gave the user the ability to work around this problem.

When the parameter c in the bump function is taken to be non-zero, then the weight function will form a ring about the point p (fig. 7). If c > r0, there will be an area in the center of the ring, centered at p, that will not move at all. Examples of the usefulness of this capability will be given below.

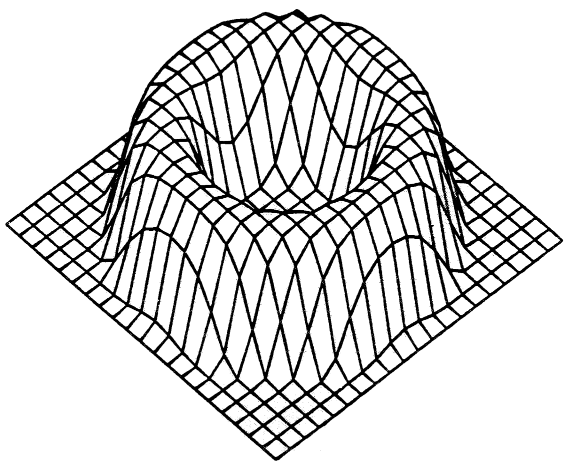


Fig 7: A translation of a portion of the surface defined by a bump function with a non-zero value of c. The surface is grabbed in the center of the ring.

Note that the above discussion trivially extends to a parameterized surface of any dimension in a space of any dimension. Let γ be a map from an n-dimensional parameter space to an m-dimensional space (usually we take $n \leq m$). T and T'(x) would then map from m-dimensional space to m-dimensional space, and if they are linear transformations they would be m x m matrices. The bump weight function would then be a function of the distance in the n-dimensional parameter space. In this way the method described here can be used to manipulate volumetric objects.

Note also that the definition of T'(x) can be generalized to any weight function. The bump function is treated specially because its parameters correspond to those aspects of the manipulation over which we desire the most control, namely the size of the region of the surface effected.

2.3: Surfaces With Topologies Different From the Square.

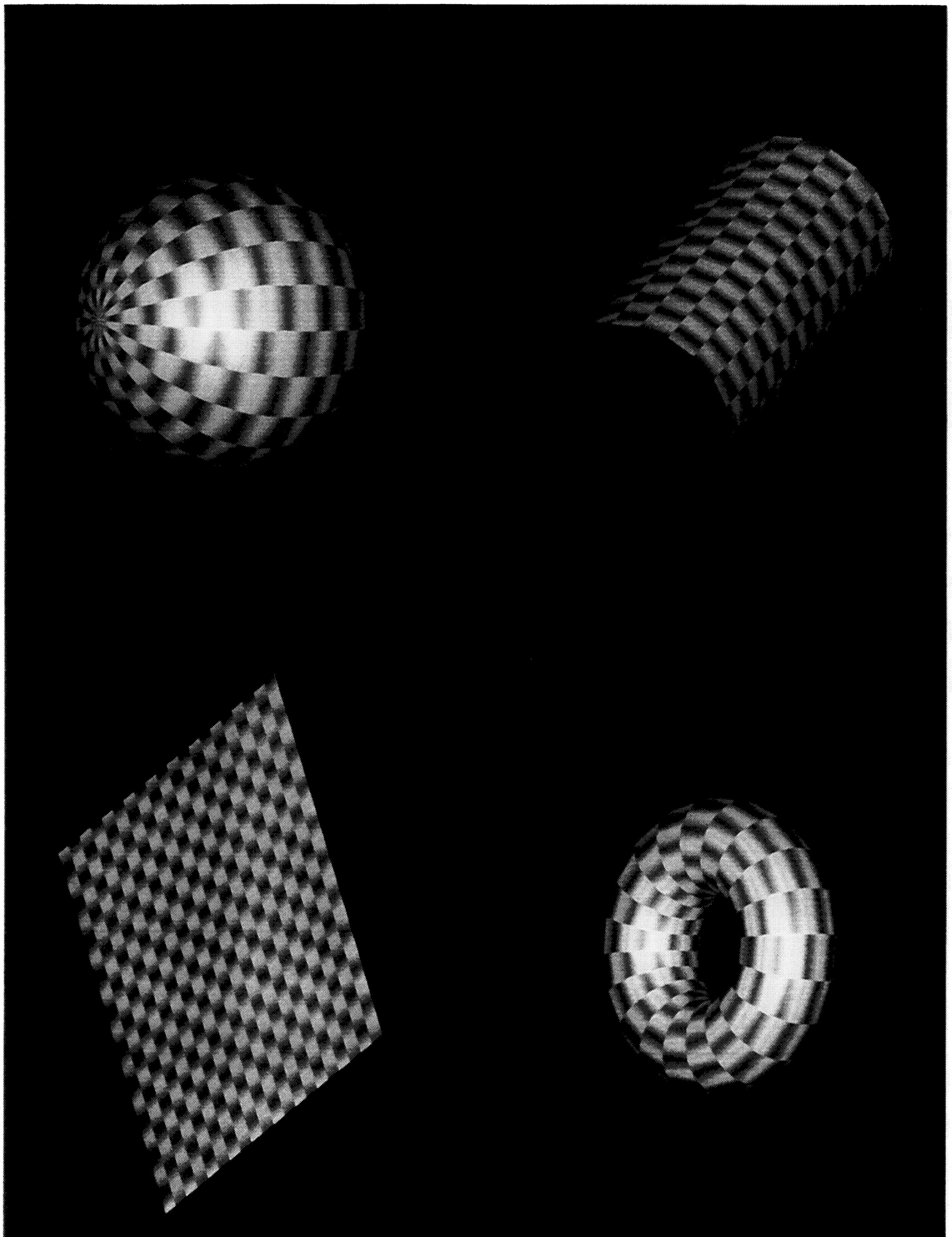
The topology of a surface is the way that the surface is connected. In the case of polygonal surfaces, the topology is determined by how the vertices of that surface are connected. There are parameterized surfaces that, while being the image a square, are topologically different from a square. Examples include the sphere, cylinder, and torus (fig. 8). Consider the sphere. The usual definition of the sphere as a parameterization of the square is obtained by identifying two opposite edges (say the edges parallel to the vertical axis), and identifying all points on each of the other edges (parallel to the horizontal axis). (Identifying points means that they are mapped to exactly the same coordinates in three-dimensional space.) This is much like the latitude/longitude system on the Earth. Using the above definition of distance in the bump function, grabbing the corner (say at (0, 0) in parameter space) and moving it will cause only those points around the image of that corner on the sphere to move. This will cause that part of the surface to be disconnected from the image of the corner (at (0, 1)) with which it should be identified, and the surface will become a sphere with a hole in it. If we want our manipulation to preserve the topology of the surface, this cannot be allowed to happen.

A simple solution is to define the distance between vertices used in the bump function to be the three-dimensional distance between vertices in the undeformed surface rather than in the parameter space. In the case of a planar initial surface, this definition of distance will be equivalent to that given above. In the case of surfaces with different topologies, however, this definition of the surface will give very different results. Points that are close together in the undeformed surface will be treated as close together in the deformed surface. This will cause points that are identified to move in identical ways, and so stay identified. This will assure that the topology of the surface is preserved.

3: Example Implementation

The system described above was implemented based on parameterized two-dimensional surfaces in three dimensions. As the surfaces are parameterzied, the vertices on the surface are defined by three functions (called components of the vertex) of two parameters (called parameters of the vertex). Thus each vertex has a pair of parameters and three components. The connectedness of the surface is defined by the ordering of the parameters, and in fact the coordinates of a vertex are stored in arrays indexed by the parameters of that vertex. This simplifies many tasks, such as finding the parameters of a vertex given its coordinates via a search of the coordinate arrays. When the user selects a vertex to be the 'pickup point' on the surface, the parameters of that point are determined in this manner.

Fig 8: Examples of two-dimensional surfaces with different topologies. Clockwise from upper left: plane; sphere; cylinder; torus.



In an implementation of the theoretical structure described above, three logically independent tasks have to be performed, given a grab point on the surface and a transformation to be applied at that point:

- Define the shape of the bump weighting function on the surface as a function of distance from the grab point
- Compute of the weighting function centered at that point for all vertices on the surface
- Compute the weighted transformation of each vertex in the surface.

Externally, the user specifies the shape of the bump function, selects a grab point to center the bump, and indicates a transformation by moving the grab point in some allowed way. The shape of the bump function is determined by the values of the parameters c, r0, and r1 defined in section 2.1.

3.1 Controlling the Shape of the Bump Weight Function

The interface for the control of the shape of the basic bump function is given by drawing a graph of the current bump function on the screen (fig 9). Handles representing the parameters c, r0, and r1 are superimposed on the graph. The user can 'click and drag' these handles to change the values of the corresponding parameters, and as the bump is redefined the graph is redrawn, giving the user feedback on the new shape of the bump.

The user must be given some indication of the strength of the bump on the surface. This feedback is particularly important in light of the fact that the bump function on the surface will be distorted as the surface is distorted, as described in section 2.2. Even without this problem, however, the possibility of exotic bump shapes (such as rings and radial modulations) requires that the user have good feedback as to exactly where on the surface the bump function will cause vertices to move. In the current implementation, this problem was solved by coloring the vertices on the surface according to the value of the bump. The color of each vertex is defined to be an interpolation between two colors, with the value of the bump function at that vertex as the interpolation parameter (fig 9).

Other weight functions beyond the bump function will need appropriate interfaces. In some sense, it is the interface that determines which weighting functions are useful, and in fact it is its relative ease of definition that makes the bump function so nice. The radial modifications to the bump function described in section 3.2 will need such an interface. One possibility would be a version of this system that manipulates one-dimensional surfaces in two dimensions, with appropriate constraints.

3.2 Computing the Bump Weight Function

Once the user has indicated a vertex p, which is considered the grabbed vertex, the parameters of p, (ps, pt) are determined. Then for all values of the parameters (s, t) for which there are vertices, the value of the bump function are computed as bump(r, c, r0, r1), where r is the euclidean distance from (ps, pt) to (s, t) = $\sqrt{(ps-s)^2 + (pt-t)^2}$. The value of bump(r, c, r0, r1) is stored in an array bval[s][t] (with s and t appropriately converted to indices), which corresponds to the vertex with parameters (s, t).

When the topology of the surface is non-trivial, as described in section 2.3, then some additional structure is required. On startup, the vertex components of the parameterized surface prior to any deformation are stored in the array of 3D vectors init[s][t]. Then when the grab point p is selected as described above, the bump function at all other vertices of the surface are computed as bump(r, c, r0, r1) where now r is the euclidean distance in three dimensions from init[ps][pt] to init[s][t]. In this way, if two vertices coincide on the initial surface, they will be transformed identically and so will always coincide.

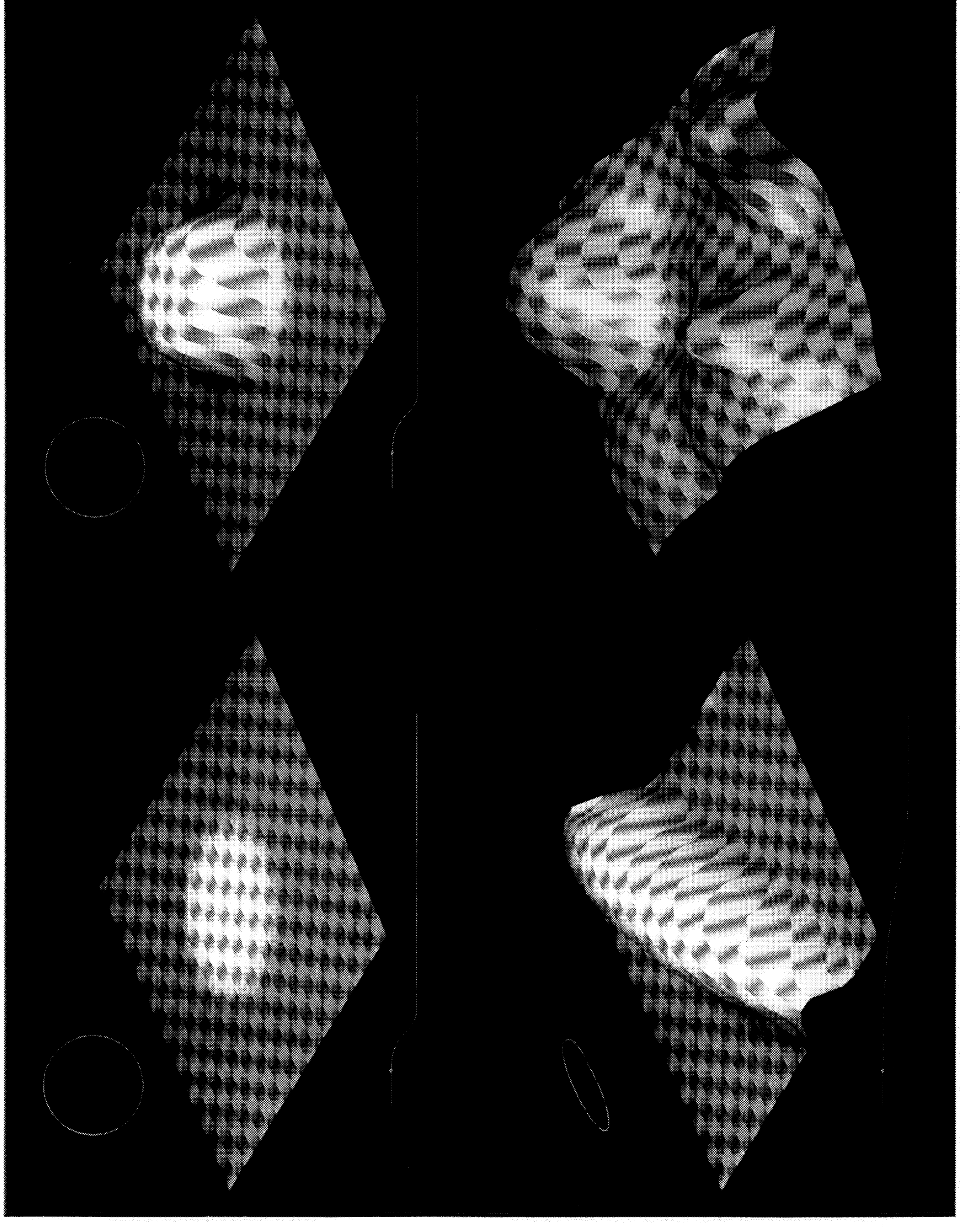
In addition to the control over the shape of the bump function described above, an ability to enrich the shape of the bump was implemented by replacing the r computed in the above paragraphs by r/f(θ), where f(θ) is some (preferably periodic) function, and θ is an angle parameter defined as arctan(ds/dt) where ds = ps - t, dt = pt - t. In this way irregular, radial variations in the bump can be easily implemented. In the author's implementation, f(θ) is a simple ellipse with variable eccentricity and phase (fig. 9).

3.3 Transforming the Vertices

The transformations that are supported in this implementation are translation, rotation, and scaling. These transformations are controlled with mouse and keyboard combination commands.

When the user indicates that a vertex p (with components (px, py, pz)) on the surface is to be grabbed, the value of the bump function centered at p is computed at each vertex of the surface as described in section 3.1. Then, by moving the input device, the user indicates the translation (tx, ty, tz), rotation **R**, and scale transformation (scalex, scaley, scalez) that are to be applied to the surface (fig. 10). The transformation at each vertex v with parameters (s, t) is weighted by the value in bval[s][t] at that vertex as follows:

Fig 9: The user's surface manipulation environment. In each figure, the line drawing under the surface is a graph of the bump function as a function of distance. The colored dots are the handles that the user moves to control parameters of the bump. The circle in the upper left allows the specification of radial distortions in the bump. The white area on the surface is the region where the bump function is non-zero. Upper left: The plane prior to manipulation. Note the cursor surrounded by white area. Upper Right: The surface after translation upwards weighted by the bump function indicated by the white area in the upper left figure. Lower left: a radially distorted bump weighting a translation upwards. Note from the graph below the figure that the bump is now very broad. The circle in the upper left of the figure has become an ellipse. The white area is now elliptical on the surface, so an elliptical hump is lifted by the transformation. Lower Right: An example of many translations and rotations weighted by many differently shaped bump functions, with the bump editing tools turned off.



$$T'(v) = \mathrm{bval}[s][t] \begin{pmatrix} \mathrm{scalex} & 0 & 0 \\ 0 & \mathrm{scaley} & 0 \\ 0 & 0 & \mathrm{scalez} \end{pmatrix} \mathbf{R} + \begin{pmatrix} 1-\mathrm{bval}[s][t] & 0 & 0 \\ 0 & 1-\mathrm{bval}[s][t] & 0 \\ 0 & 0 & 1-\mathrm{bval}[s][t] \end{pmatrix}$$

This transformation is then applied to the current components of vertex v = (vx, vy, vz) as

$$\begin{pmatrix} vx' \\ vy' \\ vz' \end{pmatrix} = T'(v) \begin{pmatrix} vx-px \\ vy-py \\ vz-pz \end{pmatrix} + \begin{pmatrix} px+tx*\mathrm{bval}[s][t] \\ py+ty*\mathrm{bval}[s][t] \\ pz+tz*\mathrm{bval}[s][t] \end{pmatrix}$$

Thus the transformations are applied to each vertex centered around p. In particular, the rotations and the scaling take p as the origin.

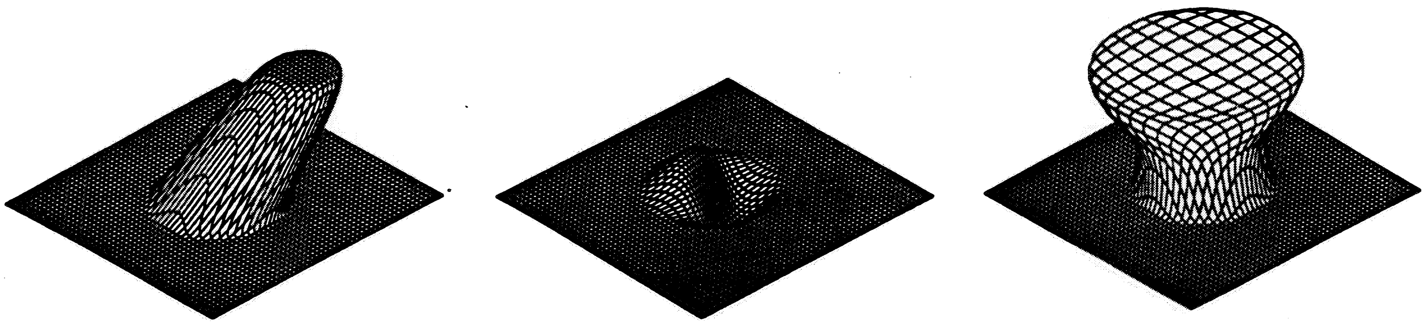


Fig 10: Three basic transformations: translation, rotation, and scaling. The scaling is performed after a translation.

3.4 Optimization and Performance

There are two computationally intensive aspects of this implementation of this system:

- Computation of the value of the bump function at each vertex. This involves computing square roots, exponentials, and (to enable radial modifications) arc tangents. These must be performed for every vertex on the surface, unless special constraints are imposed on the bump function (such as bump(x, c, r0, r1) = 0 for x > some value). They need to be done every time the user changes the shape of the bump (so the vertices effected by the bump can be displayed) or selects a new grab point. While these actions can be treated as discrete events, the ability to change the bump shape and move the grab point in real time is desirable so that the effect on the surface can be seen.
- Computation of the vertex transformation and transforming the vertex components for each vertex on the surface. Where the weight function is zero, the vertex transformation is the identity, so optimization would include checking for this case.

On an SGI iris 4D340/VGX running on a single processor, all of the above tasks ran at at least 10 frames/sec on a surface of 20 x 20 vertices, not including rendering time. As the number of vertices in the surface increased, the performance of the system dropped to about 2 frames/sec when recomputing the bump function with a surface of 100 x 100 vertices.

4: Future Work

There have been several issues raised in the development of this system. With respect to the existing implementation, the outstanding problem is the definition of distance on the surface. The methods described in section 2.2 above were motivated by the desire to keep the manipulation local with respect to the surface, and not with respect to the way that surface is embedded in three-dimensional space. A fast way to compute the distance along the surface in the three-dimensional space would be a better solution. This would require something like geodesic distance on a surface of arbitrary shape. If the surface is parameterized and is topologically the same as the square, one approach is the following: find those vertices whose parameters are close to the straight line between the points of interest in the parameter space and sum the euclidean distance between their components. While this is not exactly the same as the geodesic on the surface, this will give a sense of distance on the surface without the problems encountered above. Implementing this idea in real time is non-trivial, however. Also, in the case of non-trivial topologies such as spheres, straight lines will in general not correspond to the geodesics on the surface.

Until such a definition of distance on a surface is developed, perhaps there is some alternate definition of distance that can be developed, using the definition in section 2.2 weighted by some distance in three dimensions.

The user interface for specifying the 'grab point' on the surface and the transformation manipulating the surface can be further developed. Manipulations typically involve rotations and translations, and the user would be greatly aided by perceiving the surface as a three dimensional object. Thus it would be desirable to implement this system in the virtual environment interfaces that have been recently developed [5, 6, 12, 3].

Another interesting direction is the exploration of weight functions that are more general than the bump function. Certainly, a surface manipulation tool kit will have many choices for the weight function. The main difficulty to overcome is the development of the user interface to allow the definition of general weight functions.

For a topology visualization tool kit, several features beyond the current system are required. A visualization tool for surgery and handlebody theory would require, besides the direct manipulation of surfaces, the ability to attach surfaces together and cut surfaces apart. This is a problem for both the internal representation of the surfaces and the user interface for specifying the attach and cut points. It will also be necessary to control the rendering in sophisticated ways will be necessary, so that the user can see the interior of complicated, folded surfaces. A method of constraining the motion and detecting intersections of the surface would permit the investigation of categories of surface manipulation other than those which permit intersections.

Acknowledgements

The Author would like to thank his colleagues in the Advanced Research Office of the NAS division at NASA Ames for their many helpful comments and discussions, particularly Sam Uselton, Creon Levit, Mike Yamasaki, Al Globus, Kyra Lowther, and Horst Simon. Thanks also to Jeff Hultquist for help with Postscript and John Krystynak for enthusiasm. Special appreciation goes to Profs. Andrew Hanson and Louis Kauffman for being sources of inspiration and for their very warm encouragement.

References

1 Barr, A., Global and Local Deformations of Solid Primitives, *Computer Graphics,* Vol 17, #3, July 1984, pp 21-30

2 Bartels, R., Beatty, J, and Barsky, B., *An Introduction to Splines for use in Computer Graphics and Geometric Modeling*, Morgan Kaufmann, Los Altos, CA. 1987

3 Bryson, S. and Levit, C., *A Virtual Environment for the Exploration of Three-Dimensional Steady Flows*, RNR Technical Report #RNR-90-013, 1990, to appear in *Proceedings of the 1991 SPIE conference on Stereoscopic Displays and Applications*, San Jose, Ca. 1991

4 Coquillart, S., Extended Free-Form Deformation: A Sculpting Tool for 3D Geometric Modeling, *Computer Graphics,* Vol 24, #4, July 1990, pp 187-193

5 Fisher, S. et. al., Virtual Environment Interface Workstations, *Proceedings of the Human Factors Society 32nd Annual Meeting*, Anaheim, Ca. 1988

6 Fisher, S., Virtual Environments, Personal Simulation and Telepresence, *Implementing and Interacting with Real Time Microworlds*, Course Notes, Vol. 29, SIGGRAPH 1989

7 Foley, J., vanDam, A., Feiner, S., and Hughes, J., *Computer Graphics: Principles and Practice 2nd Ed.*, Addison-Wesley, Reading, MA. 1990

8 Forsey, D., and Bartels, R., Hierarchical B-Spline Refinement, *Computer Graphics,* Vol 22, #4, August 1988, pp 205-212

9 Francis, G., *A Topological Picturebook,* Springer-Verlag, New York, NY. 1987

10 Guillemin, V. and Pollack, A., *Differential Topology,* Prentice-Hall, Englewood Cliffs, NJ., 1974

11 Kirby, R., *The Topology of 4-Manifolds* , Lecture Notes in Mathematics #1374, Springer-Verlag, New York, 1989

12 MacDowall, I., Bolas, M., Pieper, S., Fisher, S. and Humphries, J., Implementation and Integration of a Counterbalanced CRT-based Stereoscopic Display for Interactive Viewpoint Control in Virtual Environment Applications, *Proceedings of the 1990 SPIE Conference on Stereoscopic Displays and Applications*, Santa Clara, Ca. 1990

13 Milnor, J., *Lectures on the h-Cobordism Theorem*, Mathematical Notes #1, Princeton University Press, Princeton, NJ.,1965

14 Parent, R., A System for Sculpting 3-D Data, *Computer Graphics,* Vol 11, #2, July 1977, pp 138-147

15 Sederberg, T. and Parry, S., Free-Form Deformation of Solid Geometric Models, *Computer Graphics,* Vol 20, #4, July 1986, pp 151-160

16 Terzopoulos, D., Platt, J., Barr, A. and Fleischer, K., Elastically Deformable Models, *Computer Graphics,* Vol 21, #4, July 1987, pp 205-214

17 Waters, K. and Wang, S., A 3D Interactive Physically-Based Micro World, *Extracting Meaning from Complex Data: Processing, Display, Interaction*, SPIE Vol. 1295 1990